COSC 311 PROGRAMMING PROJECT #1 Random access File I/O to implement hashing.

Distributed: 10/10/2017 **Due**: 10/30/2017

You will implement open addressing hashing using linear probing on the hard disk (in a file).

Record format

Each element of the hash table contains a record. Each record is formatted as follows:

10 chars (`name` – a String or a char[])
4 bytes (`id` – `int`; `id` is autoincrement, start at 1)
1 byte (meta -- data information byte: dirty bit, empty bit)
The data are laid out on the record as shown above.

Note! You may use an object to store the data, but the name field must occupy 10 chars.

Note!! It does not matter what the size of an input String is. You must pad or truncate to get 10 chars.

Thus, a record for a Person contains:

```
String  name;
int     id;
```

"Meta-data"

The meaning of the byte that stores the dirty bit is:

0b0000	there is no data present
0b0001	dirty — the data is invalid. I.e., this is a tombstone
0b0100	clean data — there is data present and the data is valid
0b0101	dirty data – there is data present, but the data is dirty.

Note! Only the bottom 4 bits of the byte is shown. The top four bits are, obviously, all 0.

The meta-data is information useful in the data structure (the hash table). It is not derived from any information about the Person.

The basic hash table

The hash table is a random access file comprising records. You must create the initial hash table in the file by setting up 16 records: *The hash table is initially empty and is size 16*.

Use Java Object's `hash()` function for the hash function. You will hash the String data to obtain the hash value. The record will be inserted to the file as given by the hash value.

Each record stored in the hash table will include the data associated with a Person, PLUS the meta data (one byte).

Operations on the hash table

You will insert and delete several records. When the hash table reaches 50% full, you must
pause reading input and applying operations,
create a new hash table doubled in size,
rehash the old data,
delete the old table.
After every insertion, check to see if your table has exceeded capacity (50%). Only valid data counts toward load capacity.

Note: you may determine the amount of valid data in the table by either:
1. Keeping track of the number of valid data elements in an int located at offset 0 in the file, or
2. Stepping through the table and counting the number of valid data elements.

Deletions will not cause contraction of the hash table, not even if the table is empty.

The operations are contained in a char stream file named `input.dat`. There are three kinds of operations in that file:
input (String)
delete (String)
printTable()

Input operation
`input( String str)`
(1) Read the String `str` from charstream `input.data`
(2) Generated the id. The new id is autoincremented (starting at 0).
(3) Output "Input `str - id`" to the console (obviously using the value of `str` and `id` in the output statement)
(4) Modify the String `str` to make it 20 bytes by truncating or padding as necessary (see the format method in the String classNote, the final byte is 0x04 (data present, not dirty).
(5) hash the `name` field to find the hash value. Then modify the hash value to find the seek location.
(6) Seek to the computed location.
(7) Read the entire record at the current seek position
(8) If the record location is marked 'dirty' 0x01 or 0x05, or marked 'empty' 0x00,
then insert the data (name, id, and `meta = b0100)`
else repeat linear probe until data can be inserted.
(9) After the record been inserted, check the table capacity.

If the table is over capacity:
Output "Table size `n` is overcapacity. Rehashing" (obviously, give the current value of `n`)
Rehash table
Delete any old file that will no longer be used.

Delete operation

`delete ( String str )`

(1) Read the String `str`

(2) Output "Delete `str`" to the console. Give the current value of `str` in the output

(3) Hash the data, compute the seek location

(4) Use linear probe by reading records, testing for value match or testing for dirty bit until either: data is found

or: data is not present (probed to empty record).

(5) Read the entire record in. Output "Deleted `str id`". Give values of `str` and `id`.

(6) Modify the meta data on the hash table to reflect dirty data (`b0101`)

PrintTable operation

(1) Determine the size of the file in bytes and in number or records. Output the size(s) of the file.

(2) Position the file cursor at the start of the hash table.

(3) On a separate line, output the String data, followed by id, followed by the data information byte, in the order as recorded in the file. The data information byte must be given as a bit string. If the record is empty, output blanks for the String data, followed by blanks for id, followed by the data information byte.

(4) Repeat (3) for the entire file.

Testing the data information byte:

Suppose you have put the data information byte into the unsigned byte variable named `info`.

- The data is dirty: `info && 0x01`
- The data is present and clean: `!dirty(info) && (info && 0x04)`
- The record is empty: `info == 0x00`

Setting the data information byte

Suppose you have put the data information byte into the unsigned byte variable named `info`.

- Set the dirty bit: `info = info || 0x01`
- Clear the dirty bit: `info = info && 0xFE`
- Set the data present bit: `info = info || 0x04`
- Clear the data present bit: `info = info && 0xFB`

Random access file I/O

See the description of the Java class: RandomAccessFile:
http://docs.oracle.com/javase/7/docs/api/java/io/RandomAccessFile.html

A super elementary example, with annoying advertisement, is given here:
https://www.java-tips.org/java-se-tips-100019/18-java-io/1998-how-to-use-random-access-file.html

Another example is given here: https://examples.javacodegeeks.com/core-java/io/randomaccessfile/java-randomaccessfile-example/

See course home page for links to another elementary example (including truncating/padding String data).

Turn in:

Hard copy of code
UML showing classes and relationships.
Sample run on input.dat data

Grade:

Program works as specified
Reasonable documentation of the code
UML
Style
Elegance